Redovisning av Studiecirkel/Kurs i Exekveringstidsanalys

Andreas Ermedahl†
Jan Gustafsson‡

† Department of Computer Systems, Uppsala University
Box 325, S-751 05 Uppsala, Sweden
email: ebbe@docs.uu.se

‡ Mälardalens högskola
Box 883, S-721 23 Västerås, Sweden
email: jan.gustafsson@mdh.se

Innehåll

1 Inledning

Under våren 1996 har en studiecirkel genomförts inom ramen för ASTEC/RT. Syftet har varit att studera state-of-the-art inom analys av WCET (Worst Case Execution Time) i programvara för realtidssystem. Värd för studiecirkeln har varit DoCS på Uppsala Universitet. Deltagare i kursen har kommit från DoCS, Mälardalens Högskola samt IAR. För doktoranderna bland deltagarna har studiecirkeln genomförts som en doktorandkurs på 2 poäng.

1.1 Deltagare

Deltagande personer i WCET studiecirkeln var:

Namn	**Företag/Organisation**	**Email**
Hans Hansson	DoCS	hansh@docs.uu.se
Andreas Ermedahl	DoCS	ebbe@docs.uu.se
Mikael Sjödin	DoCS	mic@docs.uu.se
Wang Yi	DoCS	yi@docs.uu.se
Paul Pettersson	DoCS	paupet@docs.uu.se
Jan Gustafsson	MdH	jan.gustafsson@mdh.se
Olle Landström	IAR Systems AB	olle.landstrom@iar.se
Anders Pikas	IAR Systems AB	anders.pikas@iar.se
Anders Berg	IAR Systems AB	anders.berg@iar.se

2 Syfte och målsättning

2.1 Motivering

Realtidssystem är system där både det funktionella och det temporala beteendet hos systemet är väsentligt. "A real-time system is a system where the correctness depends not only on the results of computations, but also on the time at which the result is produced" [Sta88].

För att kunna garantera tidsbeteendet hos ett realtidssystem måste exekveringstiden för programvaran vara känd (kunna beräknas eller mätas) i förväg. Schemaläggningsteorin för realtidssystem kräver detta, från den klassiska artikeln av Liu och Leyland från 1973 [LL73] och framåt.

Då de flesta program har en varierande exekveringstid beroende på indata och systemets tillstånd, är det WCET (Worst Case Execution Time) man vill bestämma. För program med viss komplexitet kan det vara det svårt att finna indata som tvingar programmet att exekvera den tidsmässigt längsta vägen. Man använder då någon form av statisk analys som ur programkoden beräknar WCET. Det är dessa metoder vi studerat i studiecirkeln.

Det finns även andra tillfällen då man vill känna till WCET, t ex för att kunna dimensionera hårdvara eller då man optimerar program.

2.2 Målsättning

Följande målsättning nämndes för studien:

- klassificera området.
- vad behövs?
- vad är redan löst?
- vad återstår?
- vad är svårt?

3 Uppläggning av studiecirkel

Under våren har sju stycken studiemöten genomförts (96-01-23, 96-02-12, 96-03-14, 96-04-02, 96-04-19, 96-05-28 samt 96-09-18). Till varje studiemöte har ett par deltagare läst in sig på några artiklar som de presenterat för övriga deltagare.

Tema för de olika träffarna har varit:

96-01-23 Inledning, Timing schema

96-02-12 Linjär programmering

96-03-14 Översikt, Börjessons arbete, Symbolisk exekvering, Verktygsstöd

96-04-02 Analys av modern hårdvara (cache, pipelines etc)

96-04-19 Analys av modern hårdvara (cache, pipelines etc)

96-05-28 Symbolisk exekvering, Analys av loopar

96-09-18 Abstrakt interpretering, Constraint solving och Reachability analysis

4 Undersökta områden

Nedan ges en översikt av områden och litteratur som vi fördjupat oss i under kursen.

4.1 Översikt

En bra översiktsartikel inom området är [Cha94a]. Den innehåller information om områden som:

- statisk analys

- högnivåanalys
- annoteringar
- lågnivåanalys
- kompilatorns kodgenerering
- effekter av modern hårdvara som t. ex. cache, pipelines, DMA
- mikroanalys

4.2 Högnivåanalys

Analys av högnivåkoden skrivet i något programmeringsspråk som t.ex. C eller Ada.

Timing schema:
Analys av högnivåkoden som går ut på att varje instruktion ges ett uttryck på hur lång tid den maximalt respektive minimalt kan ta att utföra. Tidsuttryck för mer sammansatta satser fås sedan genom att rekursivt lägga ihop tidsuttrycken för de i satsen ingående instruktionerna [PK89], [PS91]. Har nackdelen att hårdvara som pipeline, cache inte kan tas med i analysen samt att loopar måste annoteras (se nedan) för begränsa antalet varv i loopen.

Symbolisk exekvering:
Metod för att exekvera ett program mot ett symboliskt minne. Genom att till programs (och funktioners) inputvariabler ge abstrakta värden kan, efter en symbolisk exekvering av programmet, villkor genereras som bestämmer hur exekveringsvägen (och således WCET) kommer att se ut beroende på input. Man kan också genom att identifiera semantiskt beroende mellan vägar hitta s.k. "omöjliga vägar", och därmed minska överskattning av WCET. Används vanligen för att bevisa vissa egenskaper hos program [HK76] [B9̈6], [KE85]. Metoden har också under senare år börjat användas i WCET-analys [CBW94b], [Alt96]. Kräver dock mycket annoteringar av programmeraren, medför mycket komplex och tidsödande analys av speciellt loopar, samt hanterar bara högnivåanalys. De uttryck som genereras kan ofta vara väldigt stora och komplicerade.

Annoteringar:
Om programmeraren annoterar sin kod om hur programmet kommer att exekveras, t.ex. för att ta bort omöjliga vägar, kan en närmare uppskattning av WCET uppnås. Denna analys kan både ske på högnivåkod samt på assemblernivå [PK89], [PV91], [PK95], [Par93], [M+89], [LM95], [B9̈6]. Kräver att programmeraren alltid klarar av att annotera koden på ett korrekt sätt vilket ofta kan vara svårt.

Linjär programmering (LP):
Programkoden omvandlas till en flödesgraf där tider till instruktioner givits enligt timing schema modellen. WCET kan sedan räknas ut genom att omvandla flödesgrafen till ett linjärt uttryck som sedan skall maximeras [LM95], [PK95].

Modellen är lovande men har problemet att den, precis som timing schema, ej kan inkludera olika hårdvarumodeller och att loopar måste annoteras.

4.3 Lågnivåanalys

Analys av assembler- eller mikrokod tillsammans med en mer eller mindre utförlig hårdvarumodell.

Problemet med att inkludera cache och pipelines i modellen är att analysen måste ske på låg nivå. Man måste kunna förutsäga när en cache-miss resp. träff kommer att kunna ske och hur väl instruktioner kommer att kunna få plats i pipen. Olika modeller för att inkludera cache i WCET-analysen finns.

Cache analys:
Teknik för att grovt gruppera instruktionsanrop i förväntat cache-beteende beskrivs i [AMW94], [MWH94]. Detta ger dock lätt en stor överestimering av WCET.

[LL94] undersöker cache-beteende i loopar. Utnyttjar det faktum att om man analyserar de k första looparna m.a.p. WCET och cache så finns det inga k st senare konsekutiva loopar i samma konstruktion som kan ge högre WCET. Modellen hanterar dock inte datacache eller pipelines.

Cache och pipeline analys:
I [HBW94] ges en teknik att omvandla maskininstruktioner till mikroinstruktioner och analysera dessa m.a.p. cache och pipeline. Denna analys blir dock väldigt beroende på måldator och det är svårt att beskriva samband mellan satser, t.ex. i loopar.

[L$^+$95] har en modell där både pipelines, instruktionscache och datacache kan inkluderas i analysen. Analyserar assemblerkod och ger varje instruktion en reservationstabell över vilka resurser som den kommer att använda m.a.p. datorns klockcykler. Genom att för följder av instruktioner matcha motsvarande block mot varandra kan man hantera både pipeline- och cachebeteenden. Metoder för att minska tillståndsexplosionen vid loopar ges. Den modell och analys av hårdvaran som vi bedömt som mest lovande.

4.4 Verktygsstöd

En modell där programmeraren kan välja hela eller delar av sin högnivåkod och se motsvarande lågnivåkod presenteras i [KW95]. Till lågnivåkoden ges också motsvarande WCET framräknat m.h.a. en timing schema modell. Modellen är mycket trevlig men blir väldigt beroende av kompilator samt känslig för kompilatoroptimeringar. Vissa typer av optimeringar klaras dock.

5 Uppnådda mål och slutsatser

Kommentarer till de ursprungliga målen:

- **Klassificera området.**
  Vi har gjort ett försök till klassificering i vår tankekarta (se sektion 8). Vi har en dessutom sammanställt en referenslista med relevanta artiklar inom ämnesområdet (se sektion 9).

- **Vad behövs?**
  Det behövs verktyg för att med ett minimum av annoteringar i programkoden beräkna WCET (och BCET, Best Case Execution Time). Överreserveringen för WCET skall vara så liten som möjligt. Det är viktigt att verktyget har ett bra användargränssnitt.

  Vi ser ett tydligt behov av att göra högnivåanalysen före lågnivåanalysen. Detta för att kunna utnyttja kunskap om programmets semantik (beteende) i lågnivåanalysen. Kopplingen mellan högnivåkod och lågnivåkod ligger i kompilatorn. Analysen av WCET måste därför ha kännedom om hur kompilatorn arbetar, t.ex. med avseende på optimeringar. I lågnivåanalysen måste man ha en bra modell av hårdvaran för att kunna minimera överskattningen av WCET.

  Högnivåanalysen måste klara:

  - begränsa antalet varv i loopar
  - rekursion
  - identifiera omöjliga vägar (eller andra beroenden mellan vägar)
  - beskriva beroenden mellan indata och vägar

  Lågnivåanalysen måste klara:

  - optimerad kod
  - dynamisk minneshantering
  - DMA
  - cache-minne
  - pipelines

- **Vad är redan löst?**
  Problemet med att beräkna WCET för ett program (dock oftast med stor överskattning) är i princip löst, med "timing schema" [PK89], [PS91], LP-baserade metoder [LM95], [PK95] eller symbolisk exekvering [CBW94b], [Alt96].

  Ett antal forskare har gett sig på delproblem och mer eller mindre bra löst dessa. Se sektion 4 ovan.

- **Vad återstår?**
  Det finns två stora brister i de flesta av de befintliga metoderna:

1. De kräver annoteringar av programmeraren
2. De tar inte hänsyn till effekter av modern hårdvara.

Ingen forskare har heller försökt att samtidigt lösa alla problem, utan man har endast angripit delproblem.

Det saknas också kunskap om hur användargränssnittet mot ett WCET-verktyg borde se ut.

6 Fortsatt arbete

Områden och idéer för fortsatt arbete är:

- Göra enkät bland företag (via IAR) om vilka egenskaper de skulle vilja ha hos ett WCET-verktyg. Tanken här att fånga in de problemområden som upplevs releventa av industrin.

- Undvika annotering, istället ge varning om sådana behövs. Kompilatorn bör först automatiskt försöka hitta omöjliga vägar och maximalt antal varv i loopar. Först om den misslyckas med detta skall programmeraren behöva annotera sitt program. Metoder för att klara denna analys kan vara någon form av semantisk analys (t.ex. abstrakt interpretering eller symbolisk exekvering).

- Bättre analys av loopar. Ofta tillbringar program stor del av sin tid i loopar, speciellt i de inre looparna i nästlade loopar. Om man kan minimera antal varv i värsta fallet kan effektiv minskning av OF (överreserveringsfaktorn) erhållas. På samma sätt kan analys av alternativ av olika längd i loopar ge minskning av OF. Tanken är att undersöka vilka konstruktioner som kan tillåtas i ett programmeringsspråk för att man skall kunna beräkna antalet varv i loopar. Här kan teorier från kompilatorteori för loopar eventuellt komma till användning.

- Fördjupa oss mer i symbolisk exekvering och abstrakt interpretering. Tekniken med att m h a symbolisk exekvering bestämma WCET av program har bara använts inom två, för oss kända, fall [CBW94b], [Alt96]. Genom att kombinera denna teknik med abstrakt interpretering tror vi att flera intressanta metoder och verktyg skulle kunna tas fram.

- Göra back-end mot IARs kompilator. En kompilator samlar på sig väldigt mycket information när den omvandlar högnivåkoden till lågnivåkod. Genom att använda den informationen inom WCET analys skulle nya idéer kunna genereras. Man kan också tänka sig att utöka den information som kompilatorn samlar på sig med speciell WCET analys information. Här erbjuder samarbetet med IAR Systems ett lysande tillfälle att omsätta idéer i praktiken genom att ha de har den kompetens som krävs inom kompilatorteori samt kompilatorer att testa på.

7 DoCS, MdH:s samt IAR:s roller

DoCS vid Uppsala universitet och IAR Systems är samarbetspartners inom ASTEC-RT. Samarbetet har bland annat utmynnat i ett examensjobb, [B̈96], där en C-kompilator från IAR Systems utökades, m.h.a. timing schema- och annoteringsmetoder, för att beräkna WCET av programkoden.

Med hjälp av studiecirkeln har problemområden identifierats där framtida samarbete mellan DoCS, MdH och IAR System kan vara givande. IAR kan här bidra med mer tillämpade frågeställningar, erfarenhet av pogramutvecklares önskemål samt djup kompetens om en verklig produkt. DoCS och MdH kan bidra med forskningsresurser och teoretisk kompetens inom områden som t.ex. programanalys och kompilatorteknik.

8 Översiktskarta

Se medföljande bilaga.

9 Referenslista

[Alt96] P. Altenbernd. On the false path problem in hard real-time programs. In *Proceedings of the Eight Euromicro Workshop on Real-Time Systems*, pages 102–107, june 1996.

[AMW94] R. Arnold, F. Mueller, and D. Whalley. Bounding worst-case instruction cache performance. In *Real-Time Systems Symposium, Puerto-Rico, IEEE Press*, pages 172–181, 1994.

[B̈96] H. Börjesson. Incorporating worst case execution time in a commercial c-compiler. Master's thesis, DoCS 95/69, Dept. of Computer Systems, Uppsala, 1996. MSc Thesis.

[BJ95] S. J. Bharrat and K. Jaffay. Predicting worst case execution times on a pipelined risc processor. Technical report, , apr 1995.

[CBW94a] R. Chapman, A. Burns, and A. Wellings. Combining static worst-case timing analysis and program proof. *The Journal of Real-Time Systems, NA*, 1994.

[CBW94b] R. Chapman, A. Burns, and A. Wellings. Integrated program proof and worst-case timing analysis of spark ada. In *ACM Sigplan Workshop on Language, Compiler and Tool Support for Real-Time Systems*, june 1994.

[Cha94a] R. Chapman. Program timing analysis. Dependable Computing System Center, University of York, may 1994.

[Cha94b] R. Chapman. Worst-case timing analysis via finding longest paths in spark ada basic-paths graphs. 1994.

[For92a] C. Forsyth. Implementation of the worst-case execution analyser. *York Software Engineering Ltd*, may 1992.

[For92b] C. Forsyth. Using the worst-case execution analyser. *York Software Engineering Ltd*, july 1992.

[Gus94] J. Gustafsson. Calculating of execution times in object-oriented real-time software - a study focused on realtimetalk. Damek, KTH Stockholm, feb 1994. Lic. thesis.

[Gus95] J. Gustafsson. Calculating of execution times in realtimetalk - an object-oriented language for real-time. In *International Workshop on Object-Oriented Real-Time Dependable Systems,Dana Point, CA, USA.*, oct 1995.

[HBW94] M.R. Harmon, T.P. Baker, and D.B. Whalley. A retargetable technique for predicting execution time of code segments. *The Journal of Real-Time Systems*, 7, 1994.

[HK76] S. L. Hantler and J. C. King. An introduction to proving the correctness of programs. *Computing Surveys*, 8(3):331–353, sept 1976.

[HL95] T-Y. Huang and J. Liu. Predicting the worst-case exec. time of the concurrent exec. of instructions and cycle-stealing dma i/o oper. In *ACM Workshop on Lang., Comp. and Tools for RTS*, may 1995.

[KE85] R. A. Kemmerer and S. T. Eckmann. Unisex: a unix-based symbolic executor for pascal. *Software-practise and Expreience*, 15(5):439–458, may 1985.

[KL91a] K.B. Kenny and K-J Lin. Building flexible real-time systems using the flex language. *IEEE Computer*, may 1991.

[KL91b] K.B. Kenny and K-J Lin. Measuring and analyzing real-time performance. *IEEE Software*, sept 1991.

[KS86] E. Kligerman and A.D. Stoyenko. Real-time euclid: A language for reliable real-time systems. *IEEE Transactions on Software Engineering*, sept 1986.

[KW95] L. Ko and D. Whalley. Supporting user-friendly analysis of timing constraints. In *ACM Workshop on Lang., Comp. and Tools for RTS*, may 1995.

[L$^+$95] S. S. Lim et al. An accurate worst case timing analysis for risc processors. *IEEE Trans. on Software Engineering*, page 593, july 1995.

[LL73] C. Liu and J. Leyland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM*, pages 46 – 61, jan 1973.

[LL94] J.-C. Liu and H.-J. Lee. Deterministic upperbounds of the worst-case execution times of cached programs. In *Real-Time Systems Symposium, Puerto-Rico, IEEE Press*, pages 182–191, 1994.

[LM95] Y-T Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *ACM Workshop on Lang., Comp. and Tools for RTS*, may 1995.

[M+89] A. Mok et al. Evaluating tight execution time bounds of programs by annotations. In *Proc. of the 6th IEEE Workshop on Real-Time Operating Systems and SW, Pittsburgh*, pages 74–80, may 1989.

[MWH94] F. Mueller, D.B. Whalley, and M.R. Harmon. Predicting instruction cache behaviour. In *ACM Sigplan Workshop on Language, Compiler and Tool Support for Real-Time Systems*, june 1994.

[NR95] K. Nilsen and B. Rygg. Worst-case execution time analysis on modern processors. In *ACM Workshop on Lang., Comp. and Tools for RTS*, may 1995.

[Par93] C.Y. Park. Predicting program execution times by analyzing static and dynamic program paths. *The Journal of Real-Time System*, 5:31–62, 1993.

[PK89] P. Puschner and C. Koza. Calculating the maximum execution time of real-time programs. *The Journal of Real-Time Systems*, pages 159–176, 1989.

[PK95] P. Puschner and C. Koza. Computing maximum task execution times with linear programming techniques. Technical report, Report, Techn. Univ., Inst. für Technische Informatik, Vienna, may 1995.

[PS91] C.Y. Park and A. C. Shaw. Experiments with a program timing tool based on a source-level timing schema. *IEEE Computer*, may 1991.

[PS93] P. Puschner and A. Schedl. A tool for the computation of worst case task execution times. In *Proc. of the 5:th Euromicro Workshop on Real-Time Systems*, june 1993.

[PV91] P. Puschner and A. Vrchoticky. An assessment of task execution time analysis. Technical report, , 1991.

[SCR91] A.D. Stoyenko, C.Hamacher, and R.Holt. Analyzing hard-real-time programs for guaranteed schedulability. *IEEE Transactions on Software Engineering*, aug 1991.

[Sha91] A. C. Shaw. Towards a timing semantics for programming languages. *Found. of RT Computing, Kluwer*, pages 217–250, 1991. ISBN 0-7923-9167.

[Sta88] J. A. Stancovic. Misconceptions about real-time computing. *IEEE Computer*, 21:10–19, oct 1988.

[Vor92] K. H. Vortman. A timing analyzer for dedos. Master's thesis, Eindhoven Univ. of Techn., Dept. of Math. and Computing Science, aug 1992.

[Vrc94] A. Vrchticky. The basis for static execution time prediction. Technical report, Techn. Univ., Inst. für Technische Informatik, Vienna, apr 1994.

[W+93] Göran Wall et al. A source-level performance analysis tool for real-time programs. In *ART Real-Time Conference*, aug 1993.

[ZBN93] N. Zhang, A. Burns, , and M. Nicholson. Pipelined processors and worst case execution times. *Journal of Real-Time Systems*, 5:319–343, 1993.